Автор – Сычев Святослав, группа ИУ7-21

Изучение способов интеграции языков C/C++ и Python

Оглавление:

1. Введение
2. Способы интеграции
3. Условная задача
4. Решение на Python
5. Решение с помощью Ctypes
6. Решение с помощью Cython (без непосредственного использования C)
7. Остальные методы интеграции
8. Сравнение скоростей работы методов
9. Вывод

Введение

Python – удобная технология для разработки различных типов программных проектов, однако иногда Python в силу своей универсальности, динамической типизации, автоуправления памятью и многих других, без сомнения удобных, особенностей, проигрывает в скорости при использовании «тяжелых» ресурсоемких алгоритмов.   
  
Многие модули и библиотеки Python написаны на C, как следствие этого возникает возможность создавать собственные модули с использованием C/C++ и выносить в них ресурсоемкие части программного продукта.

Способы интеграции

Существует несколько альтернативных способов интеграции, часть из них была предусмотрена самими разработчиками Python, часть создали сторонние разработки. Вот основные способы:

* Ручное написание интерфейса модуля
* Пакет distutils
* Пакет Ctypes
* Библиотека Boost.Python
* Проект Cython
* Проект SWIG

Конечным результатом использования каждого из этих способов в большинстве случаев будет библиотека, которую можно будет подключить и использовать в стандартном коде Python.

Условная задача

Одно из наиболее эффективных направлений использования интеграции C/C++ и Python – ресурсоемкие задачи. Поэтому в рамках данной работы была разработана условная задача, требующая серьезных затрат ресурсов компьютера.

Работа с большими массивами данных – достаточно ресурсоемкий процесс, особенно, когда приходиться проводить большое число операций.

Теория графов, и задачи поиска оптимальных путей на них зачастую предполагают наличие матрицы, и выполнения большого числа вычислений.

Так как целью данной работы является изучение способов интеграции C/C++ и Python, а не написание оптимального решения достаточно сложной задачи, нам будет достаточно простой, с программной точки зрения, и условной, точки зрения реального применения задачи.

Задача:

Некоторая логистическая компания решила оптимизировать процесс выбора пути своих поездов по сети железных дорог, расположенных в достаточно опасной зоне. Часто в этой зоне случаются обвалы, или иные стихийные бедствия, поэтому некоторые пути выходят из строя.

Система датчиков собирает данные об обрывах путей, и отправляет их на главный сервер. Информация об активных путях и времени проезда по ним храниться в виде ориентированного графа, в котором вершины - станции, ребра - Ж/Д пути.

Необходимо написать программу, которая по заданной станции отправления найдет кратчайшее время пути до всех остальных станций.

Граф храниться в файле:  
1 строка – количество вершин  
2 строка – количество ребер   
далее построчно перечислены ребра в виде i j val – ребо из i в j весом val

Для решения данной задачи будет применен алгоритм Форда-Беллмана, с использованием матрицы смежности графа.

Идея алгоритма:  
1) Рассмотреть каждое ребро i-j и если оно может улучшить время, запомнить это время в массив времен   
2) Рассмотрение ребер повторить N-1 раз, где N – количество вершин.

Алгоритм будет реализован максимально просто, решение будет не оптимальным, однако, его оптимизация усложнит алгоритм и не уменьшит сложность алгоритма в разы, так что, учитывая задачу данной работы, простой реализации будет достаточно.

Решение на Python

Первое решение – это решение на «чистом» Python3.4, без использования каких либо дополнительных модулей. Далее приведен листинг алгоритма:



INF – константа условной бесконечности, выше нее время пути быть не может. Она также обозначает отсутствие ребра в матрице смежности, и отсутствие возможности попасть на станцию, если встретиться в соответствующем элементе массива времен.

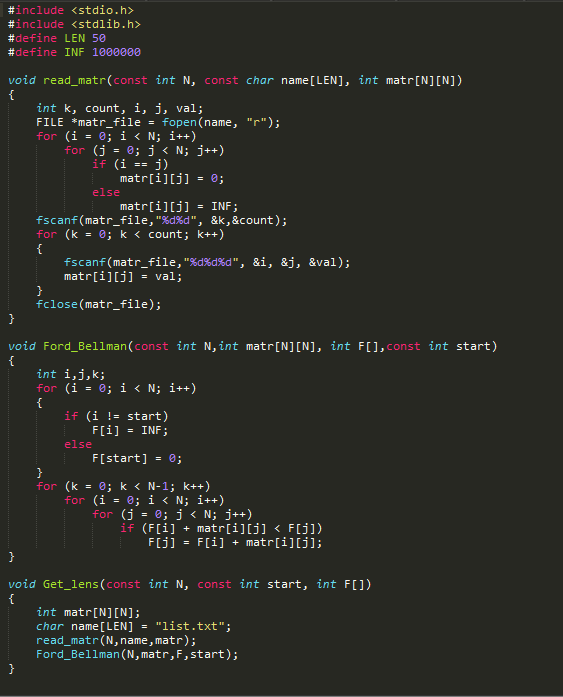
Решение с помощью Ctypes

В этом решении нам уже потребуется программировать на С.

Краткое изложение решения:

* Пишем библиотеку на С
* Компилируем ее
* С помощью Ctypes подключаем ее к Python и используем функции из этой библиотеки.

Далее приведен код библиотеки на C:

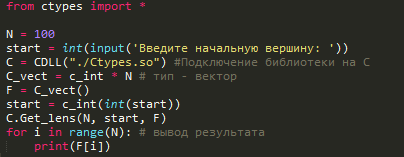


Для компиляции был использован компилятор gcc. Команда для командной строки:

команда 1.png

Результатом компиляции является библиотека **Ctypes.so**

Далее полученная библиотека импортируется в программу на Python, в которой нам остается, только задать соответствующий тип массива, используя Ctypes, и вызвать функцию Get\_lens, передав ей этот массив.  
  
Листинг программы на Python:



Стоит отметить, что в данной работе рассмотрено только частное применение Ctypes, его можно использовать даже без создания библиотеки на С, а также он позволяет создавать гораздо более серьезные библиотеки, содержащие классы, идентичные классам Python, экземпляры которых могут без проблем передаваться между библиотекой и программой на Python.

Решение с помощью Cython

Cython – отдельный программный продукт, позволяющий компилировать код Python, также он позволяет интегрировать в этот код некоторые функции из библиотек С.   
  
В данной работе будет рассмотрен пример по компиляции «чистого» кода Python при помощи Cython, без добавления функций на С, так как в Cython без проблем импортируются функции из базовых библиотек С, однако в ходе выполнения работы при попытке создания сторонней библиотеки, и импортирования функций из нее возникли проблемы, было принято решение рассмотреть на сколько компиляция «чистого» кода ускорит Python.

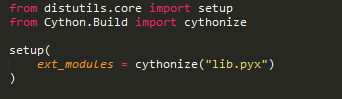
Общий принцип работы Cython:

* Создается файл-pyx на языке Cython
* Создается конфигурационный файл – setup.py
* Далее выполняется компиляция:  
  Сначала Cython компилирует .pyx-файл в .с-файл, после чего компилятор C компилирует .с-файл в .so-файл, который можно импортировать, как обычный модуль Python

Листинг .pyx-файла:



Листинг конфигурационного файла:



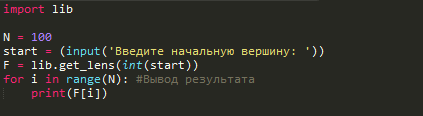
Далее этот Проводим компиляцию в командной строке:

Screenshot_4.png

Примечание – использован компилятор mingw32 (свободно распространяемый)

Результатом компиляции будет библиотека **lib.so**

Далее подключаем эту библиотеку к программе на Python, и вызываем функцию get\_lens



Остальные методы интеграции

В данной работе остальные методы будут представлены обзорно, так как они оказались довольно сложны в установке и настройке (3 дня рабочего времени было потрачено на установку и настройку SWIG и Boost.Python, однако ни один из проектов так и не дал положительных результатов).

*SWIG*

SWIG - [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) инструмент для [связывания](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B2%D1%8F%D0%B7%D1%8B%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D1%8F%D0%B7%D1%8B%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)&action=edit&redlink=1) [программ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) и [библиотек](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), написанных на [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))/[C++](https://ru.wikipedia.org/wiki/C%2B%2B), с [интерпретируемыми](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80), такими как Python, Perl, Ruby, или [компилируемыми](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) – Java, C#, языками.

Связывание происходит посредством создания специального .i-файла на специальном языке SWIG, после чего SWIG автоматически генерирует исходный код на C, после компиляции которого, можно получить библиотеку.

*Boost.Python*

Boost представляет собой **набор** из нескольких десятков разнообразных инструментов от **различных** разработчиков, иногда очень слабо связанных между собой.

Инструмент для Python - Boost Python Library, или сокращенно Boost.Python

Принцип работы Boost.Python заключается в следующем:

* Пишется исходный код библиотеки на C (на С++)
* Создается общий заголовочный файл
* Создается Файл интерфейса Boost (на С++)
* После чего модуль собирается при помощи специальных средств Boost, или вручную.

В итоге получается библиотека для Python.

К сожалению, Boost – устаревший продукт, так как его разработка завершена к 2003 году, корректность его работы с Python 3 под большим вопросом.

*Ручное написание интерфейса модуля и пакет distutils*

Этот метод подразумевает прописывание всех интерфейсов вручную, что требует глубокого изучения Python\C API, для получения достойного результата.

Пакет distutils немного упрощает этот процесс, позволяя писать файл интерфейса не на C, а на Python, однако в этом случае также требуется знание Python\C API, для написания корректно работающей библиотеки.

Сравнение скоростей работы методов

Далее приведены результаты замеров среднего времени реализаций, на чистом Python, Python + Ctypes и Python + Cython. Для более точного замера времени производиться многократный запуск, после чего берется среднее время работы.

Графы в каждом замере для всех реализаций одинаковы

Время приведено в миллисекундах.

|  |  |  |  |
| --- | --- | --- | --- |
| N – вершин | Python3.4 | Python3.4 + Ctypes | Python3.4 + Cython |
| 100 | 324.13 | 5.45 | 191.43 |
| 200 | 2535.14 | 23.47 | 1346.36 |
| 300 | 8721.04 | 55.04 | 4666.32 |
| 400 | 20267.42 | 115.08 | 11135.89 |
| 500 | 39041.78 | 198.86 | 21955.66 |

Примечание:

Для N = 400 и 500 реализация на Python3.4 была запущена по 1 разу, для экономии времени в виду явной длительности выполнения (погрешность будет мала)

Для создания графов была написана специальная программа на языке Python.

Вывод

Наиболее эффективной в рамках данной работы оказалась реализация на Сtypes.

Если необходимо ускорить код Python, вполне возможно использование Cython - даже без подключения библиотек и функций С/С++, в нашем случае это ускорило работу примерно в 2 раза.

Так как в ходе работы мною были рассмотрены и остальные методы (хоть мне и не удалось их применить, из-за возникших проблем с установкой и настройкой) могу сказать, что для создания сторонних библиотек на C, самым простым способом можно считать Ctypes, так как он прост в установке и не требует особо серьезных знаний С/С++ для написания простых библиотек.